

I - Introduction.....	3
II - Contrôles de ruban utilisant des images.....	3
III - Utilisation d'une image standard : attribut ImageMso.....	3
IV - Chargement initial : callback loadImage.....	5
V - Chargement à la demande : callback getImage.....	6
VI - Image dynamique sur un bouton bascule.....	8
VII - Intégrer les images à l'application : utilisation d'un champ pièce-jointe.....	10
VIII - Utilisation de GdiPlus : transparence des images et format png.....	12
VIII-A - Chargement de fichiers image avec transparence.....	12
VIII-B - Chargement d'image avec transparence depuis un champ pièce-jointe.....	13
IX - Transparence des images et format ico.....	14
IX-A - Chargement de fichiers icônes avec transparence.....	14
IX-B - Chargement d'icônes avec transparence depuis un champ pièce-jointe.....	15
X - Conclusion.....	16
XI - Les téléchargements.....	16



## I - Introduction


Dans la **version 2007** d'Office, les barres de menus ont été remplacées par **le ruban**.

La programmation de ce ruban se fait en **XML**.

Les images affichées sur les contrôles du ruban peuvent être soit :

- des images standards intégrées à Office.
- des images personnalisées.

Nous allons voir les différentes techniques pour gérer ces images.

 *Pour créer plus facilement le XML de vos rubans, utilisez l'**assistant ruban**.*

 *Pour mieux comprendre la programmation des rubans, consultez ces tutoriels :*

- **Programmez et personnalisez le ruban de vos applications Access 2007.**
- **La personnalisation du ruban sous Excel 2007.**
- **Personnalisation du ruban: Les fonctions d'appel Callbacks.**
- **Comment personnaliser le Ruban de Word 2007.**

## II - Contrôles de ruban utilisant des images

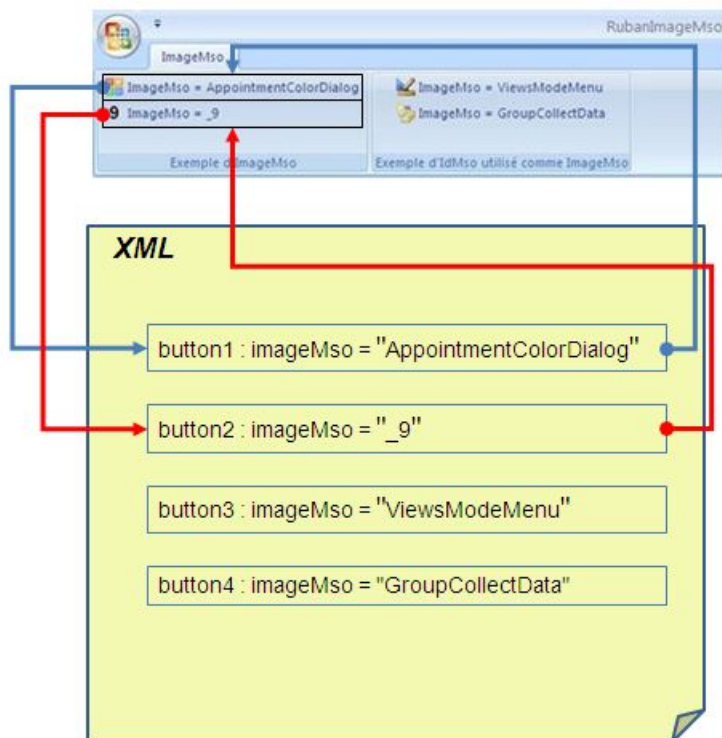
Les types de contrôles du ruban qui affichent une image sont :

Type de contrôle	Description
<b>button</b>	Bouton.
<b>comboBox</b>	Liste déroulante avec une zone de texte d'édition.
<b>dropDown</b>	Liste déroulante sans zone de texte d'édition.
<b>dynamicMenu</b>	Menu dynamique
<b>editBox</b>	Zone de texte d'édition.
<b>gallery</b>	Galerie de contrôles.
<b>group</b>	Groupe contenant les contrôles.
<b>toggleButton</b>	Bouton bascule.

## III - Utilisation d'une image standard : attribut ImageMso

De nombreuses images standards sont disponibles : elles sont installées avec Office.

Pour utiliser une de ces images, définissez tout simplement l'attribut XML **imageMso** du contrôle.



La liste des valeurs possibles pour **imageMso** est disponible dans un fichier Excel sur le site de microsoft : [Liste des imageMso](#).

Notez qu'il est également possible d'utiliser une des valeurs de l'attribut **idMso** proposées dans la [Liste des idMso](#). Voici un exemple de code XML utilisant l'attribut **imageMso** :

```

Utilisation de imageMso

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab1" label="ImageMso">
        <group id="group1" label="Exemple d'ImageMso">

          <button id="button1" imageMso="AppointmentColorDialog" label="ImageMso = AppointmentColorDialog"/>
            <button id="button2" imageMso="_9" label="ImageMso = _9"/>
          </group>

          <group id="group2" label="Exemple d'IdMso utilisé comme ImageMso">

            <button label="ImageMso = AppointmentColorDialog" imageMso="ViewsModeMenu" id="button3"/>

            <button id="button4" label="ImageMso = GroupCollectData" imageMso="GroupCollectData"/>
          </group>
        </tab>
      </tabs>
    </ribbon>
  </customUI>


```

Ce code XML suffit à afficher les images.

Aucun code VBA n'est nécessaire.

Voici le ruban obtenu :



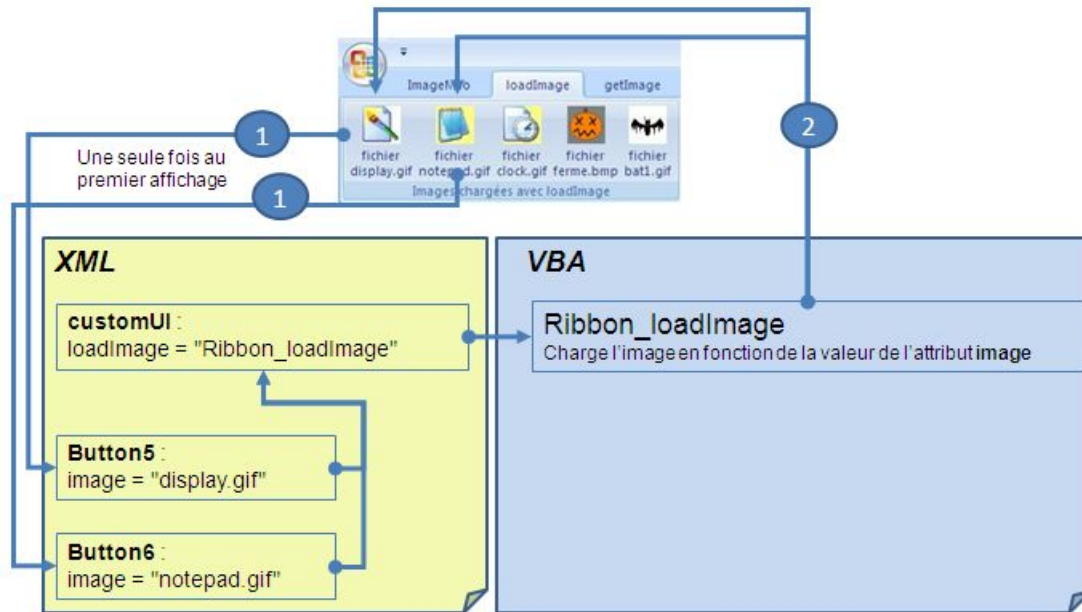
 Utilisez l'**assistant ruban** pour trouver plus facilement l'**imageMso** dont vous avez besoin.


## IV - Chargement initial : callback loadImage

Un callback est un appel à une procédure VBA.

Si vous ne savez pas comment fonctionnent les callback, je vous invite à consulter les tutoriels cités en [introduction](#). **loadImage** est un callback associé à l'élément **CustomUI**.

La procédure ainsi définie sera **exécutée une seule fois** pour chaque contrôle dont l'attribut **image** est renseigné. Elle sera ignorée en cours d'utilisation de l'application, même après un appel à **Invalidate** ou **InvalidateControl**.



 Cette attribut **loadImage** ne peut donc pas être utilisé pour des images dynamiques qui doivent changer en fonction d'événements dans la base de données.



Voici le début du code XML d'un ruban utilisant l'attribut **loadImage** :

```
Attribut loadImage
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadImage="Ribbon_loadImage">
```

La procédure VBA **Ribbon\_loadImage** sera exécutée au premier affichage de chaque contrôle de ruban dont l'attribut **image** est renseigné.

Voici comment l'écrire :

```
Procédure Ribbon_loadImage
Sub Ribbon_loadImage(imageId As String, ByRef image)
End Sub
```

La procédure nous envoie un paramètre **imageId** : c'est en fait la valeur de l'attribut **image** de l'élément dont l'image doit être chargée.

En retour, on doit modifier le paramètre **image** qui attend un objet de type **IPictureDisp**.

Un objet de type **IPictureDisp** peut être simplement créé avec la fonction VBA **LoadPicture**.

Voici un exemple de code XML utilisant l'attribut **loadImage** :

### Utilisation de loadImage

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadImage="Ribbon_loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab2" label="loadImage">
        <group id="group3" label="Images chargées avec loadImage">

<button id="button5" label="fichier display.gif" image="display.gif" size="large"/>

<button id="button6" label="fichier notepad.gif" image="notepad.gif" size="large"/>
      <button id="button7" label="fichier clock.gif" image="clock.gif" size="large"/>
      <button id="button8" label="fichier ferme.bmp" image="ferme.bmp" size="large"/>
      <button id="button9" label="fichier bat1.gif" image="bat1.gif" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>

```

Avec, dans un module VBA, le code de la fonction de chargement des images :

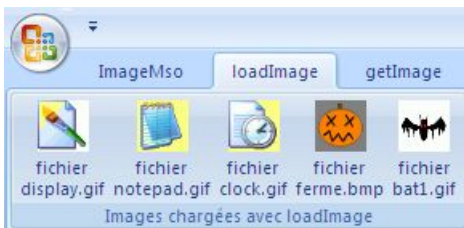
### Procédure de chargement des images

```

Sub Ribbon_loadImage(imageId As String, ByRef image)
Set image = LoadPicture(CurrentProject.Path & "\images\" & imageId)
End Sub

```

Notez que dans cet exemple, les images sont des fichiers stockés dans un sous-répertoire nommé **images**.  
Voici le ruban obtenu :



***i** La transparence des images n'a pas été conservée (les images gif utilisées sont des images avec transparence).  
Nous verrons plus loin dans cet article comment conserver la transparence des images chargées.*

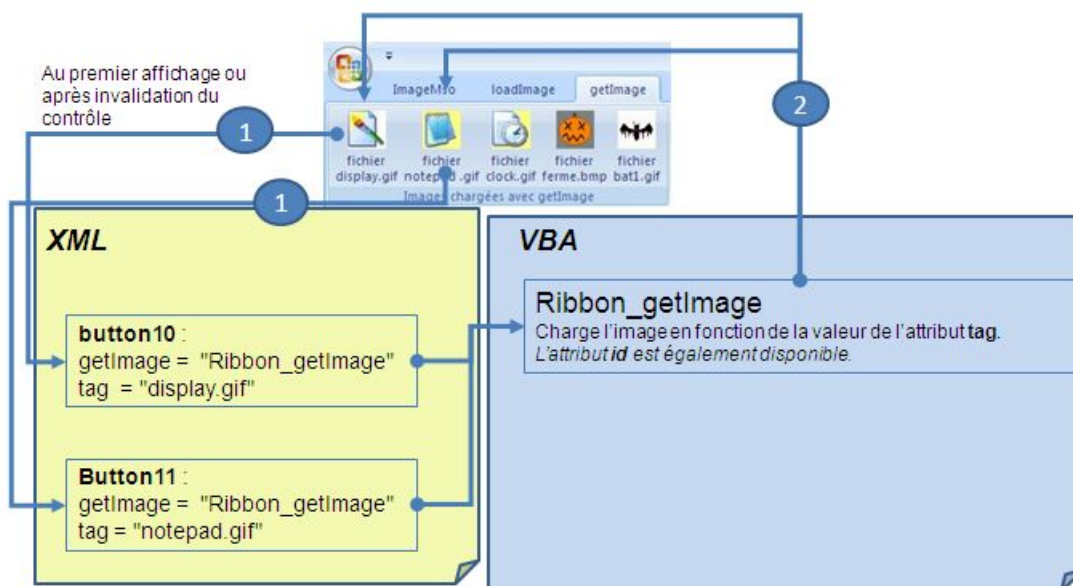
## V - Chargement à la demande : callback getImage

**getImage** est un callback associé aux éléments qui affichent une image.

***i** Pour des listes ou galeries, on trouve également le callback **getItemImage**.*

La procédure ainsi définie sera exécutée pour chaque contrôle :

- Au premier affichage du contrôle.
- Après un appel à **Invalidate** ou **InvalidateControl**.



Voici le code XML d'un bouton de ruban utilisant l'attribut **getImage** :

#### Attribut getImage

```
<button id="button9" label="fichier bat1.gif" tag="bat1.gif" getImage="Ribbon_getImage" size="large"/>
```

La procédure VBA **Ribbon\_getImage** sera exécutée au premier affichage du bouton, puis à chaque fois que le contrôle est invalidé par un appel à **Invalidate** ou **InvalidateControl**.

Voici comment écrire cette procédure :

#### Procédure Ribbon\_loadImage

```
Sub Ribbon_getImage(control As IRibbonControl, ByRef image)
End Sub
```

**⚠** Activez la référence à **Microsoft Office 12.0 Object Library** pour définir le type de données **IRibbonControl**

La procédure nous envoie un paramètre **control** : c'est l'élément dont l'image doit être chargée.

En retour on doit modifier le paramètre **image** qui attend un objet de type **IPictureDisp**.

Un objet de type **IPictureDisp** peut être simplement créé avec la fonction VBA **LoadPicture**.

Voici un exemple de code XML utilisant l'attribut **getImage** :

#### Utilisation de getImage

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab3" label="getImage">
        <group id="group4" label="Images chargées avec getImage">
          <button id="button10" label="fichier display.gif" tag="display.gif" getImage="Ribbon_getImage" size="large"/>
          <button id="button11" label="fichier notepad.gif" tag="notepad.gif" getImage="Ribbon_getImage" size="large"/>
          <button id="button12" label="fichier clock.gif" tag="clock.gif" getImage="Ribbon_getImage" size="large"/>
          <button id="button13" label="fichier ferme.bmp" tag="ferme.bmp" getImage="Ribbon_getImage" size="large"/>
          <button id="button14" label="fichier bat1.gif" tag="bat1.gif" getImage="Ribbon_getImage" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```



### Utilisation de getImage

```

        </tab>
    </tabs>
</ribbon>
</customUI>

```

Avec, dans un module VBA, le code de la fonction de chargement des images :

### Procédure de chargement des images

```

Sub Ribbon_getImage(control As IRibbonControl, ByRef image)
Set image = LoadPicture(CurrentProject.Path & "\images\" & control.Tag)
End Sub

```

Notez que dans cet exemple, les images sont des fichiers stockés dans un sous-répertoire nommé **images**.

Le nom du fichier à charger est stocké dans l'attribut **Tag** du contrôle du ruban.


Voici le ruban obtenu :



L'affichage des images est identique avec **loadImage** ou **getImage**.

La seule différence dans cet exemple est que dans un cas on utilise l'attribut **image** pour stocker le nom du fichier et dans l'autre cas on utilise l'attribut **tag**.

Dans le chapitre suivant, nous allons détailler une utilisation de **getImage** qui ne pourrait être réalisée avec **loadImage**.

 *La transparence des images n'a pas été conservée (les images gif utilisées sont des images avec transparence).*

*Nous verrons plus loin dans cet article comment conserver la transparence des images chargées.*

## VI - Image dynamique sur un bouton bascule

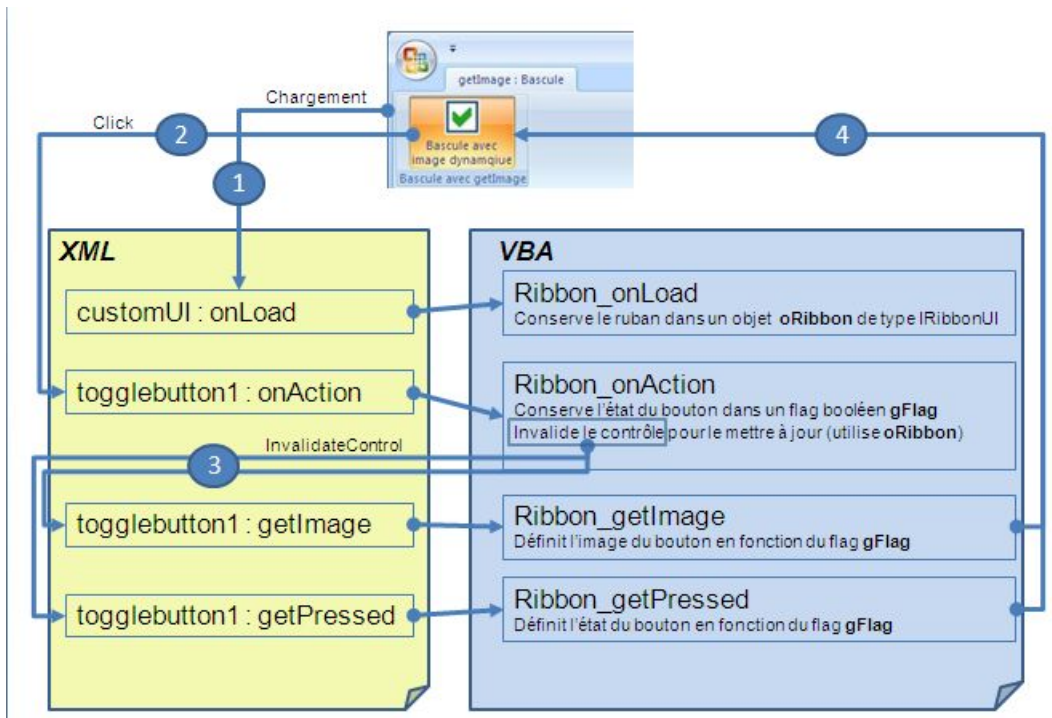
Au cours de ce chapitre, nous allons réaliser une case à cocher de grande taille.

Pour la réaliser, nous utilisons un bouton bascule dont l'image est modifiée en fonction de l'état appuyé ou non du bouton.

Pour modifier l'image dynamiquement, il faut :

- utiliser la procédure de rappel **onLoad** du ruban (élément **customUI**) afin d'obtenir un objet **IRibbonUI** représentant le ruban.
- utiliser la procédure de rappel **onAction** du bouton pour agir en VBA lors du click sur le bouton.
- utiliser l'objet ruban (**IRibbonUI**) pour invalider le bouton et ainsi déclencher la mise à jour de ce bouton.
- utiliser la procédure de rappel **getImage** du bouton bascule pour mettre à jour l'image.

Nous allons également utiliser la procédure de rappel **getPressed** du bouton pour synchroniser l'état du bouton avec une variable VBA.



Voici le code XML du ruban :

#### XML pour bouton bascule

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad="Ribbon_onLoad">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab4" label="getImage : Bascule">
        <group id="group5" label="Bascule avec getImage">
          <toggleButton id="togglebutton1" getImage="Ribbon_getImage"
            onAction="Ribbon_onAction" label="Bascule avec image dynamique"
            getPressed="Ribbon_getPressed" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Dans le code VBA nous ajoutons, en en-tête d'un module, la déclaration de l'objet ruban et du flag qui représente l'état du bouton bascule :

#### Déclarations

```
Private gFlag As Boolean
Private oRibbon As IRibbonUI
```

Ensuite, nous utilisons la procédure de rappel pour conserver l'objet ruban :

#### Définition de l'objet ruban au chargement

```
Sub Ribbon_onLoad(ribbon As IRibbonUI)
  Set oRibbon = ribbon
End Sub
```

**oRibbon** est alors défini au chargement du ruban.

Cet objet nous permet d'invalider tout ou une partie du ruban.

Lorsqu'on clique sur le bouton, la procédure **Ribbon\_onAction** est exécutée.

On récupère dans le paramètre **pressed** l'état du bouton (enfoncé ou relâché) et on le stocke dans la variable **gFlag**.



On appelle ensuite la méthode **InvalidateControl** du ruban pour déclencher la mise à jour du bouton.

#### Sur action sur le bouton bascule

```
Sub Ribbon_onAction(control As IRibbonControl, pressed As Boolean)
Select Case control.Id
    Case "togglebutton1"
        gFlag = pressed
        oRibbon.InvalidateControl control.Id
End Select
End Sub
```

Lorsqu'on invalide le bouton avec la méthode **InvalidateControl**, toutes ses procédures de rappel **get\*** sont exécutées.

La procédure **getImage** nous permet de définir l'image en fonction de l'état du bouton :

#### Mise à jour de l'image

```
Sub Ribbon_getImage(control As IRibbonControl, ByRef image)
Select Case control.Id
    Case "togglebutton1"
        If gFlag Then
            ' Image de case cochée
            Set image = LoadPicture(CurrentProject.Path & "\images\cocher.jpg")
        Else
            ' Image de case décochée
            Set image = LoadPicture(CurrentProject.Path & "\images\decocher.jpg")
        End If
End Select
End Sub
```

Et la procédure **getPressed** détermine l'état du bouton, que l'on synchronise avec notre flag **gFlag**.

#### Mise à jour de l'état

```
Sub Ribbon_getPressed(control As IRibbonControl, ByRef returnValue)
Select Case control.Id
    Case "togglebutton1"
        returnValue = gFlag
End Select
End Sub
```

Il n'est pas utile d'employer cette procédure de rappel **getPressed** si l'état du bouton est uniquement modifié par clic de l'utilisateur.

Mais en définissant cette procédure, on peut également modifier l'état du bouton de ruban par code VBA :

#### Mise à jour de l'état par code VBA

```
' Définit l'état à "cocher"
gFlag = True
' Met à jour le bouton bascule
oRibbon.InvalidateControl "togglebutton1"
```

Voilà donc un exemple d'utilisation de **getImage**.

Cet exemple n'aurait pas pu être réalisé avec la fonction **loadImage** qui n'est appelée qu'une seule fois.

## VII - Intégrer les images à l'application : utilisation d'un champ pièce-jointe

Peut-être souhaitez-vous intégrer les images du ruban dans votre fichier Access ?

Si le code XML est lui aussi **contenu dans une table de la base de données**, tout le ruban sera alors inclus dans le fichier Access.

Il n'y aura plus de fichiers externes.

Les fichiers images peuvent être intégrés dans un champ de type pièce-jointe (nouveau apparu avec Access 2007).

Créez d'abord une table nommée **TRibbonImg** avec pour champs :

- **id** : champ texte.
- **img** : pièce-jointe.

TRibbonImg	
Nom du champ	Type de données
id	Texte
img	Pièce jointe

Créez ensuite un formulaire simple **FrmRibbonImg** contenant les deux champs de la table **TRibbonImg** et ayant cette table comme source (utilisez si nécessaire l'assistant de création de formulaire).

Nommez le contrôle pièce-jointe **img**.



Ajoutez chaque image dans la table.

Utilisez un enregistrement par image, ne stockez pas plusieurs images dans le même champ pièce-jointe.

Le champ **id** est utilisé pour identifier l'image dans la table : vous pouvez décider d'y mettre un id de contrôle ou n'importe quel identifiant pourvu qu'il soit unique.

Si vous utilisez l'identifiant du contrôle, vous devrez définir une image par contrôle même si plusieurs contrôles utilisent la même image.

Une solution simple et efficace consiste à utiliser un identifiant qui sera défini dans l'attribut **tag** du contrôle de ruban (ou l'attribut **image** si on utilise **loadImage**).

Dans cet exemple, nous ajoutons des images avec comme id le nom du fichier :

TRibbonImg		
id		Ajouter un nouveau champ
bat1.gif	0(1)	
cigogne.gif	0(1)	
ferme.bmp	0(1)	
*	0(0)	



#### XML pour images chargées depuis une table

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadImage="Ribbon_loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab5" label="Image depuis un champ pièce-jointe">
        <group id="group6" label="Image depuis un champ pièce-jointe">
          <button id="button15" label="id bat1.gif" image="bat1.gif" size="large"/>
          <button id="button16" label="id cigogne.gif" image="cigogne.gif" size="large"/>
          <button id="button17" label="id ferme.bmp" image="ferme.bmp" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>

```

Il faut enfin, pour charger les images et les injecter dans le ruban :

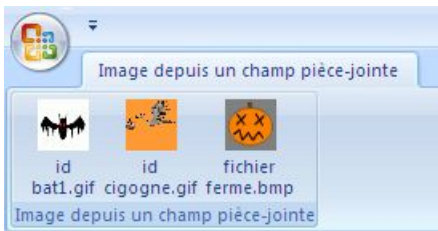
- ouvrir le formulaire positionné sur l'enregistrement désiré
- et utiliser la méthode cachée **PictureDisp** du contrôle pièce-jointe

### Procédure de chargement d'images

```

Sub Ribbon_loadImage(imageId As String, ByRef image)
On Error GoTo Gestion_Erreurs
DoCmd.OpenForm "FrmRibbonImg", , , "id=" & imageId & "", , acHidden
Set image = Forms("FrmRibbonImg").img.PictureDisp
Exit Sub
Gestion_Erreurs:
MsgBox "Image invalide : " & imageId
End Sub
    
```

Notez qu'on ouvre le formulaire en mode invisible (acHidden) car on ne souhaite pas le voir à l'écran. Voici le ruban obtenu :



*La transparence des images n'a, cette fois-ci encore, pas été conservée (les images gif utilisées sont des images avec transparence). Nous verrons plus loin dans cet article comment conserver la transparence des images chargées.*

## VIII - Utilisation de GdiPlus : transparence des images et format png

Cela ne vous a sans doute pas échappé, les méthodes décrites jusqu'ici ont deux limitations majeures :

- la transparence des images gif n'est pas conservée.
- le format png n'est pas accepté.

Le format png est un format d'image très intéressant : bonne qualité d'image, taille de fichier réduite, et prend en charge la transparence.

Il est possible d'aller au delà de ces limitations en utilisant **GdiPlus** pour charger les images.

**GdiPlus** est une librairie graphique introduite avec Windows XP.

La librairie est en téléchargement pour les systèmes d'exploitation suivants (Windows 2000; Windows 98; Windows ME; Windows NT; Windows XP; Windows Vista) :

**Lien vers la librairie en téléchargement sur Microsoft.com**

Le fichier gdiplus.dll doit être placé dans le répertoire de l'application.

Mais utiliser les fonctions de GdiPlus n'est pas très facile.

Voici donc un module de classe à utiliser dans vos applications :

**Module de classe cIRibbonImage.**

Insérez le contenu de ce fichier texte dans un module de classe (**Insertion => Module de classe**) et nommez ce module **cIRibbonImage**

### VIII-A - Chargement de fichiers image avec transparence

Le XML reste inchangé, faisons le test avec **loadImage** :

#### XML avec loadImage

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadImage="Ribbon_loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab2" label="loadImage">
        <group id="group7" label="Images chargées avec loadImage et gdiplus">
    
```

### XML avec loadImage

```

<button id="button5_gdiplus" label="fichier display.gif" image="display.gif" size="large"/>
<button id="button6_gdiplus" label="fichier notepad.gif" image="notepad.gif" size="large"/>
<button id="button7_gdiplus" label="fichier clock.gif" image="clock.gif" size="large"/>
<button id="button8_gdiplus" label="fichier ferme.bmp" image="ferme.bmp" size="large"/>
<button id="button9_gdiplus" label="fichier bat1.gif" image="bat1.gif" size="large"/>
  </group>
</tab>
</tabs>
</ribbon>
</customUI>

```

Avec, dans un module VBA, le code de la fonction de chargement des images :

### Procédure de chargement des images avec GdiPlus

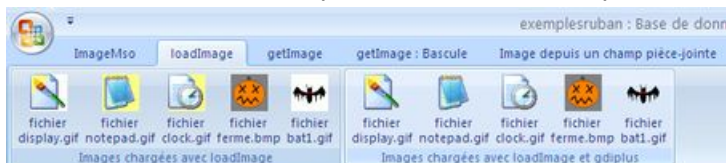
```

Sub Ribbon_loadImage(imageId As String, ByRef image)
Dim loGdi as New clRibbonImage
Set Image = loGdi.LoadFromFile(CurrentProject.Path & "\images\" & imageId)
End Sub

```

Il suffit de déclarer un objet de type **clRibbonImage** et de remplacer l'appel à **LoadPicture** par un appel à la fonction **LoadFromFile** de cet objet.

Voici le ruban obtenu, comparé au résultat obtenu précédemment avec **LoadPicture** :



La transparence est au rendez-vous!

Il est également possible de charger des fichiers png avec cette nouvelle fonction.

## VIII-B - Chargement d'image avec transparence depuis un champ pièce-jointe

Nous utilisons dans ce chapitre la même table **TRibbonImg** que celle créée précédemment.

Le XML reste inchangé, faisons le test cette fois-ci avec **getImage** :

### XML avec getImage

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab3" label="getImage">
        <group id="group8" label="Images depuis un champ pièce-jointe et GdiPlus">
          <button id="button15_plus" label="id bat1.gif" tag="bat1.gif"
            getImage="Ribbon_getImage" size="large"/>
          <button id="button16_plus" label="id cigogne.gif" tag="cigogne.gif"
            getImage="Ribbon_getImage" size="large"/>
          <button id="button17_plus" label="id ferme.bmp" tag="ferme.bmp"
            getImage="Ribbon_getImage" size="large"/>
          <button id="button18_plus" label="id ok.png" tag="ok.png"
            getImage="Ribbon_getImage" size="large"/>
          <button id="button19_plus" label="id ship.png" tag="ship.png"
            getImage="Ribbon_getImage" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>

```

Avec, dans un module VBA, le code de la fonction de chargement des images :

#### Procédure de chargement des images avec GdiPlus

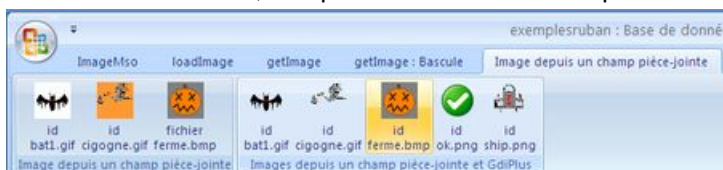
```
Public Sub Ribbon_getImage(ByVal control As IRibbonControl, ByRef image)
Dim loGdi as New clRibbonImage
Set image = loGdi.LoadFromAttachment("TRibbonImg", "img", "id=" & control.Tag & "'")
End Sub
```

Il suffit de déclarer un objet de type **clRibbonImage** et de remplacer l'appel à **LoadPicture** par un appel à la fonction **LoadFromAttachment** de cet objet.

Cette fonction **LoadFromAttachment** demande 3 paramètres :

- **pTable** : Nom de la table qui contient les images.
- **pField** : Nom du champ pièce-jointe.
- **pWhere** : Clause Where pour rechercher une image en particulier.

Voici le ruban obtenu, comparé au résultat obtenu précédemment avec un formulaire masqué :



La transparence est au rendez-vous!

Remarque : il est également possible de charger des fichiers png avec cette nouvelle fonction.

## IX - Transparence des images et format ico

Une autre solution pour obtenir de la transparence pour les images du ruban est d'utiliser le format .ico des icônes. Là encore, il faut utiliser quelques API pour charger les icônes mais gdiplus n'est pas utile ici.

Le code utilisé pour charger les fichiers ico est allégé par rapport au précédent code utilisant gdiplus, mais seul le format ICO est supporté.

Il existe de nombreux outils gratuits pour créer des fichiers ico et également des convertisseurs en ligne, par exemple **Converticon**.

Vous pouvez aussi trouver des icônes prêts à l'emploi sur internet; vérifier toutefois leur licence d'utilisation au préalable.

Voici donc un module à utiliser dans vos applications pour charger les fichiers ico:

### **Module standard ModRibbonIcon.**

Insérez le contenu de ce fichier texte dans un module standard (**Insertion => Module**) et nommez ce module **ModRibbonIcon** par exemple.

## IX-A - Chargement de fichiers icônes avec transparence

Nous avons ici 3 icônes dans un sous-répertoire *images*.

Faisons le test avec **loadImage** :

#### XML avec loadImage

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" loadImage="Ribbon_loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab4" label="Icônes">
        <group id="group9" label="Icônes depuis un fichier">
          <button id="button1_icon" label="fichier clock.ico" image="clock.ico"/>
          <button id="button2_icon" label="fichier help.ico" image="help.ico"/>
          <button id="button3_icon" label="fichier timbre.ico" image="timbre.ico"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
```

### XML avec loadImage

```
</customUI>
```

Avec, dans un module VBA, le code de la fonction de chargement des images :

#### Procédure de chargement des images avec GdiPlus

```
Sub Ribbon_loadImage(imageId As String, ByRef image)
Set image = IconLoadFromFile(CurrentProject.Path & "\images\" & imageId)
End Sub
```

Il suffit de remplacer l'appel à **LoadPicture** par un appel à la fonction **IconLoadFromFile**.  
Voici le ruban obtenu :



Notez que la fonction accepte un deuxième paramètre *pIconNumber* pour éventuellement préciser le numéro de l'icône dans le fichier.

En effet un fichier ico peut contenir plusieurs images de tailles différentes.

Dans la base de données en téléchargement, vous trouverez un exemple d'utilisation de ce paramètre.

## IX-B - Chargement d'icônes avec transparence depuis un champ pièce-jointe

Nous utilisons dans ce chapitre la même table **TRibbonImg** que celle créée précédemment.

Nous y ajoutons les 3 icônes, avec en identifiant le nom du fichier.

Faisons le test cette fois-ci avec **getImage** :

### XML avec getImage

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab4" label="Icônes">
        <group id="group10" label="Icônes depuis une pièce-jointe">
          <button id="button4_icon" label="pièce-
jointe clock.ico" tag="clock.ico" getImage="Ribbon_GetImage"/>
          <button id="button5_icon" label="pièce-
jointe help.ico" tag="help.ico" getImage="Ribbon_GetImage"/>
          <button id="button6_icon" label="pièce-
jointe timbre.ico" getImage="Ribbon_GetImage" tag="timbre.ico"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Avec, dans un module VBA, le code de la fonction de chargement des images :

#### Procédure de chargement des images avec GdiPlus

```
Public Sub Ribbon_getImage(ByVal control As IRibbonControl, ByRef image)
Set image = IconLoadFromAttachment("TRibbonImg", "img", "id='" & control.Tag & "'")
End Sub
```

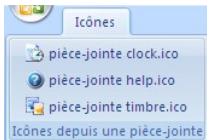
Il suffit de remplacer l'appel à **LoadPicture** par un appel à la fonction **IconLoadFromAttachment**.

Cette fonction **IconLoadFromAttachment** demande au minimum 3 paramètres :

- **pTable** : Nom de la table qui contient les images.
- **pField** : Nom du champ pièce-jointe.
- **pWhere** : Clause Where pour rechercher une image en particulier.



Voici le ruban obtenu; le résultat est identique à ce qu'on obtient avec des fichiers externes :



Notez que la fonction accepte un dernier paramètre *plconNumber* pour éventuellement préciser le numéro de l'icône dans le fichier.

En effet un fichier ico peut contenir plusieurs images de tailles différentes.

Dans la base de données en téléchargement, vous trouverez un exemple d'utilisation de ce paramètre.

On peut par exemple utiliser un numéro d'icône pour un bouton de taille *normal* et un autre pour un bouton de taille *large*.

## X - Conclusion

On a donc à notre disposition plusieurs méthodes pour gérer les images du ruban.

Il faut faire un choix en fonction de notre besoin :

- images standards ou besoin d'images personnalisées ?
- images intégrées ou fichiers externes ?
- besoin de transparence ou non ?

J'invite les plus courageux à découvrir (ou re-découvrir) les modules de classe **clGdiplus** et **clGdi32**.

(Le module **clRibbonImage** est en fait une version remaniée de **clGdiplus**.)

Ces modules de classe permettent de générer des images et de les utiliser ensuite dans le ruban grâce à leur méthode **GetPictureDisp**.

Toutes les fantaisies deviennent possibles, même d'afficher un gif animé dans le ruban grâce à GdiPlus.

## XI - Les téléchargements

**Télécharger le module de classe pour GdiPlus au format texte (secours HTTP)**

**Télécharger le module pour gérer les icônes au format texte (secours HTTP)**

**Télécharger la base d'exemples au format ACCESS 2007 (secours HTTP)**